



# COMPUTING SCIENCE

A Specification for ACMs

C. B. Jones

**TECHNICAL REPORT SERIES**

---

No. CS-TR-1360

November 2012

## **A Specification for ACMs**

**C. B. Jones**

### **Abstract**

This note proposes a simple specification for Asynchronous Communication mechanisms. In particular, it makes use of rely/guarantee conditions and the newer "possible values" notation.

## Bibliographical details

JONES, C.B.

A Specification for ACMS  
[By] Cliff B. Jones

Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1360)

### Added entries

NEWCASTLE UNIVERSITY  
Computing Science. Technical Report Series. CS-TR-1360

### Abstract

This note proposes a simple specification for Asynchronous Communication Mechanisms. In particular, it makes use of rely/guarantee conditions and the newer "possible values" notation.

### About the authors

Cliff B. Jones is currently Professor of Computing Science at Newcastle University. As well as his academic career, Cliff has spent over 20 years in industry. His 15 years in IBM saw among other things the creation –with colleagues in Vienna– of VDM which is one of the better known “formal methods”. Under Tony Hoare, Cliff wrote his doctoral thesis in two years. From Oxford, he moved directly to a chair at Manchester University where he built a world-class Formal Methods group which –among other projects– was the academic lead in the largest Software Engineering project funded by the Alvey programme (IPSE 2.5 created the “mural” (Formal Method) Support Systems theorem proving assistant). He is now applying research on formal methods to wider issues of dependability. Until 2007 his major research involvement was the five university IRC on “Dependability of Computer-Based Systems” of which he was overall Project Director. He is also PI on an EPSRC-funded project “AI4FM” and coordinates the “Methodology” strand of the EU-funded DEPLOY project. He also leads the ICT research in the ITRC Program Grant. Cliff is a Fellow of the Royal Academy of Engineering (FREng), ACM, BCS, and IET. He has been a member of IFIP Working Group 2.3 (Programming Methodology) since 1973 (and was Chair from 1987-96).

### Suggested keywords

SOFTWARE  
CONCURRENCY  
VERIFICATION  
RELY/GUARANTEE

# A specification for ACMs

Cliff B. Jones

School of Computing Science, Newcastle University, UK  
cliff.jones@ncl.ac.uk

November 16, 2012

## Abstract

This note proposes a simple specification for Asynchronous Communication Mechanisms. In particular, it makes use of rely/guarantee conditions and the newer “possible values” notation.

## 1 A specification with sequential ordering

In [JP11, Fig. 5], we wrote the specification of Asynchronous Communication Mechanisms (ACMs) using two sequentially composed sub operations for both the reader and the writer.<sup>1</sup>

*Write(v: Value)*  
**owns wr** *data-w, fresh-w*  
*start-Write(v: Value)*  
**wr** *data-w*  
**guar**  $\{1..fresh-w\} \triangleleft data-w = \{1..fresh-w\} \triangleleft \overline{data-w}$   
**post**  $data-w = \overline{data-w} \curvearrowright [v]$   
*commit-Write()*  
**wr** *fresh-w*  
**rd**  $\overline{data-w}$   
**guar**  $fresh-w \leq \overline{fresh-w}$   
**post**  $fresh-w = \text{len } data-w$

*Read(): Value*  
**owns wr** *hold-r*  
*start-Read()*  
**wr** *hold-r*  
**rd** *fresh-w*  
**rely**  $\overline{fresh-w} \leq \overline{fresh-w}$   
**post**  $hold-r \in \overline{fresh-w}$   
*end-Read(): Value*  
**rd** *data-w, hold-r*  
**rely**  $data-w(hold-r) = \overline{data-w}(hold-r)$   
**post**  $r = data-w(hold-r)$

The use of “semicolon” in specifications caused some raised eyebrows.

---

<sup>1</sup>Actually, there is a “typo” in the cited paper — **guar** of *start-Read* in the paper should be (as here) **rely**!

## 2 An alternative

We could have written:<sup>2</sup>

*Write*( $v$ : Value)  
**wr**  $\overline{data-w}, \overline{fresh-w}$   
**rely**  $\overline{fresh-w} = \overline{fresh-w} \wedge \overline{data-w} = \overline{data-w}$   
**guar**  $\overline{fresh-w} \leq \overline{fresh-w} \wedge \{1..\overline{fresh-w}\} \triangleleft \overline{data-w} = \{1..\overline{fresh-w}\} \triangleleft \overline{data-w}$   
**post**  $\overline{data-w} = \overline{data-w} \overset{\wedge}{\curvearrowright} [v] \wedge \overline{fresh-w} = \mathbf{len} \overline{data-w}$

*Read*( $r$ : Value)  
**rd**  $\overline{data-w}, \overline{fresh-w}$   
**rely**  $\overline{fresh-w} \leq \overline{fresh-w} \wedge \{1..\overline{fresh-w}\} \triangleleft \overline{data-w} = \{1..\overline{fresh-w}\} \triangleleft \overline{data-w}$   
**post**  $\exists \text{hold-}r \in \overline{fresh-w} \cdot r = \overline{data-w}(\text{hold-}r)$

The match between *guar-Write* and *rely-Read* is deceptive: for *Write* it means that  $\overline{data-w}$  must be changed before extending  $\overline{fresh-w}$ ; for *Read*,  $\overline{fresh-w}$  must be read before  $\overline{data-w}$  is accessed.

To see that the above specification has the required behaviour (customers are assumed to be fully conversant with rely/guarantee conditions and the “possible values” notation!) note the following:

- the sequential (no interleaving) use is obvious from the post conditions (alone);
- if *Read* is interrupted by (possibly many) *Write*, the post condition still shows that one of the written values is returned and the rely condition ensures that it cannot be a contaminated value;
- a little thought is required to see that, when two reads overtake a write, the second read cannot access an older value than the first;
- the *Write* process is unaffected by overtaking *Read* providing its rely condition is respected.

## 3 Observations

**How did we miss this?** My recollection is that the “possible values” notation only became clear during the steps of development.

**Revising proofs** The obvious step is simply to prove that our original specification is a refinement of the new one: this ought be straightforward. Alternatively, it doesn’t look difficult to show that [JP11, Fig. 7] is a refinement of the new specification. I guess we should also check whether there are any other simplifications.

## References

- [JP11] Cliff B. Jones and Ken G. Pierce. Elucidating concurrent algorithms via layers of abstraction and reification. *Formal Aspects of Computing*, 23(3):289–306, 2011.

---

<sup>2</sup>Here the **owns** notation has been replaced by a rely condition on *Write*.